View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Archivio istituzionale della ricerca - Università degli Studi di Udine

A SAT-based encoding of the one-pass and tree-shaped tableau system for LTL

Luca Geatti[1,2], Nicola Gigante[1], and Angelo Montanari[1]

[1] University of Udine, Italy
angelo.montanari@uniud.it, nicola.gigante@uniud.it
[2] Fondazione Bruno Kessler, Trento
lgeatti@fbk.eu

Abstract. Recently, a new one-pass and tree-shaped tableau system for LTL satisfiability checking has been proposed, with the distinguishing features of (i) being really tree-shaped, in contrast to previous graph-shaped tableau methods, and (ii) requiring only one pass to either accept or reject a branch of the tableau. Despite its simplicity, it proved itself to be effective in practice. In this paper, we provide a SAT-based encoding of such a tableau system, based on the technique of *bounded satisfiability checking*. Starting with a single-node tableau, *i.e.*, depth k of the tree-shaped tableau equal to zero, we proceed in an incremental fashion. At each iteration, the tableau rules are encoded in a Boolean formula, representing all branches of the tableau up to the current depth k. A typical downside of such bounded techniques is the effort needed to understand when to stop incrementing the bound, to guarantee the completeness of the procedure. In contrast, termination and completeness of the proposed algorithm is guaranteed without computing any upper bound to the length of candidate models, thanks to the Boolean encoding of the PRUNE rule of the original tableau system. We conclude the paper by describing a tool that implements our procedure, and comparing its performance with other state-of-the-art LTL solvers.

Keywords: tableau system · temporal logic · satisfiability · SAT.

1 Introduction

Linear Temporal Logic (LTL) is one of the most used temporal logics in formal verification. In this context, the main problem is *model checking* [9], *i.e.*, deciding whether a given specification is satisfied by a given system. However, testing a system against a valid or unsatisfiable formula can be useless at best, and dangerous at worst, and thus *sanity checking* of specifications is an important step in model-based design [27]. For this reason, the *satisfiability problem*, *i.e.*, establishing whether a formula admits any model in the first place, has been given an important amount of research effort. In addition to its application to formal verification, it also plays a role in AI systems [16, 20], *e.g.*, in planning problems.

Besides their relevance in applications, decidability and complexity of the satisfiability problem are always among the first issues to answer about a logic.

Since the first computational complexity results [25], many techniques have been devised over the last decades, with *tableau methods* being among the first methods to be developed [18, 19, 24]. In contrast to earlier tableau methods for classical logic [4, 10], that work by building a suitable derivation tree, most of these methods build a *graph* structure, whose paths represent possible evolutions of the computation, and then look for those ones that satisfy all the properties required by the formula. Recently, a *tree-shaped* tableau for LTL has been proposed by Reynolds [22], which only requires a single pass to decide whether a given branch has to be accepted or not. The smaller size of the tree with regards to the full graph structure of previous methods, and its simple rule-based tree search mechanism, led to an efficient implementation [3], a simple and fruitful parallelization [21], and modular extensions to more expressive logics [13, 14].

In this paper, we propose a satisfiability checking procedure for LTL formulae based on a *SAT encoding* of the one-pass and tree-shaped tableau by Reynolds [22]. The tableau tree is (symbolically) built in a breadth-first way, by means of Boolean formulae that encode all the tableau branches up to a given depth k, which is increased at every step. The expansion rules of the tableau system are encoded in the formulae in such a way that a successful assignment represents a branch of the tree of length k, which in turn represents a model for the original LTL formula. This breadth-first iterative deepening approach has been exploited in the past by *bounded satisfiability checking* and *bounded model checking* algorithms [7, 15], which share with us the advantage of leveraging the great progress of SAT solvers in the last decades, and the *incrementality* of such solvers.

A common drawback of existing bounded satisfiability checking methods is the difficulty in identifying when to stop the search in the case of *unsatisfiable* formulae. In order to ensure termination, either a global upper bound has to be computed in advance, which is not always possible or feasible, or some other techniques are needed to identify where the search can be stopped. In our system, termination is guaranteed by a suitable encoding of the tableau's PRUNE rule. This rule was the main novelty of Reynolds' one-pass and tree-shaped system when it was originally proposed [22], has a clean model-theoretic interpretation [13], and the important role it plays in our encoding adds up to its interesting properties. The result is a simple and complete bounded satisfiability checking procedure based on a small and much simpler SAT encoding.

We implemented the proposed procedure and encoding in a tool, called BLACK for (Bounded LTL sAtisfiability ChecKer), and we report the outcomes of an initial experimental evaluation, comparing it with other state-of-the-art tools. The results are promising, consistently improving over the tableau explicit construction.

The paper proceeds as follows. Section 2 includes a brief account of LTL and of Reynold's one-pass and tree-shaped tableau system. Section 3 shows the base encoding of the tableau rules, excepting the PRUNE rule, building a system that terminates correctly on satisfiable instances. Later, Section 4 describes and discusses the encoding of the PRUNE rule, completing the procedure. Section 5 describes the BLACK tool, together with the results of the experimental evaluation. Section 6 concludes and highlights possible future developments.

2 Preliminaries

2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) is a propositional modal logic interpreted over infinite (discrete) linear orders. Syntactically, LTL can be viewed as an extension of propositional logic with the *tomorrow* ($\mathsf{X}\,\phi$), *until* ($\alpha\,\mathcal{U}\,\beta$), and *release* ($\alpha\,\mathcal{R}\,\beta$) operators. Given a set $\Sigma = \{p, q, r, \ldots\}$ of atomic propositions, LTL formulae are inductively defined as follows:

$$\begin{aligned} \phi \coloneqq p \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 & \qquad\qquad \text{Boolean operators} \\ \mid \mathsf{X}\,\phi_1 \mid \phi_1\,\mathcal{U}\,\phi_2 \mid \phi_1\,\mathcal{R}\,\phi_2 & \qquad\qquad \text{temporal operators} \end{aligned}$$

Note that, given the disjunction and *until* operator, the conjunction and *release* ones are not necessary (in particular, $\phi_1\,\mathcal{R}\,\phi_2 \equiv \neg(\neg\phi_1\,\mathcal{U}\,\neg\phi_2)$). However, it is useful to consider them as primitive, in order to allow any LTL formula ϕ to be put into *negated normal form*, producing a linear-size equivalent formula, noted as $\mathrm{nnf}(\phi)$, such that negations appear only applied to proposition letters. Moreover, common shorthands can be defined, such as the *eventually* ($\mathsf{F}\,\phi_1 \equiv \top\,\mathcal{U}\,\phi_1$) and *always* ($\mathsf{G}\,\phi_1 \equiv \neg\,\mathsf{F}(\neg\phi_1)$) operators.

LTL formulae are interpreted over infinite *state sequences* $\overline{\sigma} = \langle\sigma_0, \sigma_1, \ldots\rangle$, with $\sigma_i \subseteq \Sigma$ for each $i \geq 0$. Given a state sequence $\overline{\sigma}$, a position $i \geq 0$, and an LTL formula ϕ, the satisfaction of ϕ by $\overline{\sigma}$ at position i, written $\overline{\sigma} \models_i \phi$, is inductively defined as follows:

1. $\overline{\sigma} \models_i p$ iff $p \in \sigma_i$
2. $\overline{\sigma} \models_i \neg\phi$ iff $\overline{\sigma} \not\models_i \phi$
3. $\overline{\sigma} \models_i \phi_1 \vee \phi_2$ iff either $\overline{\sigma} \models_i \phi_1$ or $\overline{\sigma} \models_i \phi_2$
4. $\overline{\sigma} \models_i \phi_1 \wedge \phi_2$ iff $\overline{\sigma} \models_i \phi_1$ and $\overline{\sigma} \models_i \phi_2$
5. $\overline{\sigma} \models_i \mathsf{X}\,\phi$ iff $\overline{\sigma} \models_{i+1} \phi$
6. $\overline{\sigma} \models_i \phi_1\,\mathcal{U}\,\phi_2$ iff there exists $j \geq i$ such that $\overline{\sigma} \models_j \phi_2$ and $\overline{\sigma} \models_k \phi_1$ for all $i \leq k < j$
7. $\overline{\sigma} \models_i \phi_1\,\mathcal{R}\,\phi_2$ iff for all $j \geq i$, either $\overline{\sigma} \models_j \phi_2$ or there exists $i \leq k < j$ such that $\overline{\sigma} \models_k \phi_1$.

We say that $\overline{\sigma}$ *satisfies* ϕ, written $\overline{\sigma} \models \phi$, if and only if the state sequence $\overline{\sigma}$ satisfies ϕ at its first state, *i.e.*, $\overline{\sigma} \models_0 \phi$. In this case, we say that $\overline{\sigma}$ is a *model* of ϕ.

2.2 The one-pass and tree-shaped tableau system

We now describe Reynolds' tableau system for LTL. After its original formulation in [22], the system was extended to support past operators [14] and more expressive real-time logics [13]. Here, we briefly recall its original future-only version, which is the one considered for the SAT encoding described in the next section.

The tableau for a formula ϕ is a tree where each node u is labelled by a set $\Gamma(u)$ of formulae from the closure $\mathcal{C}(\phi)$ of ϕ. At each step of the construction,

Rule	ϕ	$\Gamma_1(\phi)$	$\Gamma_2(\phi)$
DISJUNCTION	$\alpha \vee \beta$	$\{\alpha\}$	$\{\beta\}$
UNTIL	$\alpha\,\mathcal{U}\,\beta$	$\{\beta\}$	$\{\alpha, \mathsf{X}(\alpha\,\mathcal{U}\,\beta)\}$
RELEASE	$\alpha\,\mathcal{R}\,\beta$	$\{\alpha, \beta\}$	$\{\beta, \mathsf{X}(\alpha\,\mathcal{R}\,\beta)\}$
EVENTUALLY	$\mathsf{F}\,\beta$	$\{\beta\}$	$\{\mathsf{X}\,\mathsf{F}\,\beta\}$
CONJUNCTION	$\alpha \wedge \beta$	$\{\alpha, \beta\}$	
ALWAYS	$\mathsf{G}\,\alpha$	$\{\alpha, \mathsf{X}\,\mathsf{G}\,\alpha\}$	

Table 2. Tableau expansion rules. When a formula ϕ of one of the types shown in the table is found in the label Γ of a node u, one or two children u' and u'' are created with the same label as u excepting for ϕ, which is replaced, respectively, by the formulae from $\Gamma_1(\phi)$ and $\Gamma_2(\phi)$.

a set of rules is applied to each leaf node. Each rule can possibly append one or more children to the node, or either *accept* (✓) or *reject* (✗) the node. The construction continues until all leaves are either accepted or rejected, resulting into at least one accepted leaf if and only if the formula is satisfiable, with the corresponding branch representing a satisfying model for the formula. A node whose label contains only *elementary* formulae, *i.e.*, propositions or *tomorrow* operators, is called a *poised* node. At each step, the *expansion rules* are applied to any non-poised leaf node. The rules are given in Table 2. For each non-elementary formula $\psi \in \mathcal{C}(\phi)$, the corresponding expansion rule defines two sets of expanded formulas $\Gamma_1(\psi)$ and $\Gamma_2(\psi)$, with the latter possibly empty. The application of the rule to a node u adds a child u' to u such that $\Gamma(u') = \Gamma(u) \setminus \{\psi\} \cup \Gamma_1(\psi)$, and, if $\Gamma_2(\psi) \neq \emptyset$, a second child u'' such that $\Gamma(u'') = \Gamma(u) \setminus \{\psi\} \cup \Gamma_2(\psi)$.

Expansion rules are applied to non-poised nodes until a poised node is produced. Then, a number of *termination rules* are applied, to decide whether the node can be accepted, rejected, or the construction can proceed. In what follows, a formula of the type $\mathsf{X}(\alpha\,\mathcal{U}\,\beta)$ is called X-*eventuality*. Given a branch $\overline{u} = \langle u_0, \ldots, u_n \rangle$, an X-eventuality ψ is said to be *requested* in some node u_i if $\psi \in \Gamma(u_i)$, and *fulfilled* in some node u_j, with $j \geq i$, if $\beta \in \Gamma(u_j)$.

Let $\overline{u} = \langle u_0, \ldots, u_n \rangle$ be a branch with poised leaf u_n. The termination rules are the following, to be applied in the given order:

- EMPTY: If $\Gamma(u_n) = \emptyset$, then u_n is *accepted*.
- CONTRADICTION: If $\{p, \neg p\} \subseteq \Gamma(u_n)$, for some $p \in \Sigma$, then u_n is *rejected*.
- LOOP: If there is a poised node $u_i < u_n$ such that $\Gamma(u_n) = \Gamma(u_i)$, and all the X-eventualities requested in u_i are fulfilled in the nodes between u_{i+1} and u_n, then u_n is *accepted*.
- PRUNE: If there are three positions $i < j < n$, such that $\Gamma(u_i) = \Gamma(u_j) = \Gamma(u_n)$, and among the X-eventualities requested in these nodes, all those fulfilled between u_{j+1} and u_n are fulfilled between u_{i+1} and u_j as well, then u_n is *rejected*.

If the branch is neither accepted nor rejected, the construction of the branch proceeds to the next temporal step by applying the STEP rule.

STEP A child u_{n+1} is added to u_n such that $\Gamma(u_{n+1}) = \{\psi \mid \mathsf{X}\,\psi \in \Gamma(u_n)\}$.

Intuitively, given an accepted branch of the complete tableau for ϕ, the poised nodes are labelled by the formulae that hold in the states of the corresponding model for the formula. Depending on whether the branch is accepted by the EMPTY or the LOOP rule, it either corresponds to a finite (also called *loop-free*) model or to a periodic one (also called *lasso-shaped*), whose period corresponds to the segment in between the nodes that trigger the LOOP rule. If a branch is rejected, it happens either because of a logical contradiction, that triggers the $\mathsf{CONTRADICTION}$ rule, or because of the PRUNE rule, which avoids the tableau to infinitely postpone a request that is impossible to fulfil. From a model-theoretic point of view [13], the PRUNE rule allows one not to consider models that contain *redundant segments*, *i.e.*, segments that just repeat some previously done piece of work without contributing further to the satisfaction of all the pending requests. Recent work [13] provided a model-theoretic interpretation of this mechanism, showing a characterization of the discarded models.

3 SAT-based encoding of the tableau

This section describes the SAT-based encoding of Reynolds' tableau. In particular, we first describe the whole encoding apart from the PRUNE rule, which is described and discussed in detail in the next section, where the complete satisfiability checking procedure is provided.

As already pointed out, the overall structure of our procedure is similar to other *bounded satisfiability checking* approaches. At each step k, ranging from zero upwards, we produce a Boolean formula $|\phi|^k$, which represents all the accepted branches of the tableau of depth at most k. The satisfaction of such a formula witnesses the existence of an accepted branch of the tableau, which in turn proves the existence of a model for the formula. If the formula is unsatisfiable, we can proceed to the next depth level. Note that this corresponds to a symbolic breadth-first traversal of the complete tableau for ϕ. Such a procedure would be incomplete, possibly running forever on some unsatisfiable instances, without some halting criterion, which in our case is provided by the encoding of the PRUNE rule as described in Section 4. Let us now proceed with the description of the base encoding. In what follows, any LTL formula is assumed to be in *negated normal form*.

3.1 Notation

We now define some notation, useful for what follows. Let ϕ be an LTL formula (in negated normal form) over the alphabet Σ. The *closure* of ϕ is the set of formulae $\mathcal{C}(\phi)$ defined as follows:

1. $\phi \in \mathcal{C}(\phi)$;
2. if $\mathsf{X}\,\psi \in \mathcal{C}(\phi)$, then $\psi \in \mathcal{C}(\phi)$;
3. if $\psi \in \mathcal{C}(\phi)$, then $\Gamma_1(\phi) \subseteq \mathcal{C}(\phi)$ and $\Gamma_2(\phi) \subseteq \mathcal{C}(\phi)$ (as defined in Table 2).

Then, let $\mathsf{XR}(\phi) \subseteq \mathcal{C}(\phi)$ be the set of all the *tomorrow* formulae (X-*requests*) in $\mathcal{C}(\phi)$, *i.e.*, all the formulae $\mathsf{X}\,\psi \in \mathcal{C}(\phi)$, and let $\mathsf{XEV} \subseteq \mathsf{XR}(\phi)$ be the set of all the X-eventualities in $\mathcal{C}(\phi)$, *i.e.*, all the formulae $\mathsf{X}(\alpha\,\mathcal{U}\,\beta) \in \mathcal{C}(\phi)$.

The propositional encoding of the formula ϕ is defined over an extended alphabet Σ_+, which includes:

1. any proposition from the original alphabet Σ;
2. the *grounded* X-*requests*, *i.e.*, a proposition noted as ψ_G for all $\psi \in \mathsf{XR}(\phi)$;
3. a *stepped* version p^k, for any $k \in \mathbb{N}$, of all the propositions p above, with p^0 identified as p.

Some notation complements the above extended propositions. In particular, for all $\psi \in \mathcal{C}(\phi)$, we denote by ψ_G the formula obtained by replacing any $\rho \in \mathsf{XR}(\phi)$ appearing in ψ by ρ_G. Similarly, for all $\psi \in \mathcal{C}(\phi)$, we denote as ψ^k, with $k \in \mathbb{N}$, the formula obtained from ψ by replacing any proposition p with p^k. Intuitively, different stepped versions of the same proposition p are used to represent the value of p at different states. From now on, for any formula $\psi \in \mathcal{C}(\phi)$, we will write ψ_G^k as a shorthand for the formula $((\psi)_G)^k$.

Finally, we recall the definition of a simple transformation of LTL formulae which is heavily used in our encoding.

Definition 1 (Next Normal Form)**.** *An LTL formula ϕ is in* next normal form *iff every* until *or* release *subformula appears in the operand of a* tomorrow*.*

An LTL formula ϕ can be turned into its *next normal form* equivalent formula $\mathrm{xnf}(\phi)$ as follows:

1. $\mathrm{xnf}(p) \equiv p$ and $\mathrm{xnf}(\neg p) = \neg p$ for all $p \in \Sigma$;
2. $\mathrm{xnf}(\mathsf{X}\,\psi_1) \equiv \mathsf{X}\,\psi_1$ for all $\mathsf{X}\,\psi_1 \in \mathcal{C}(\phi)$;
3. $\mathrm{xnf}(\psi_1 \wedge \psi_2) \equiv \mathrm{xnf}(\psi_1) \wedge \mathrm{xnf}(\psi_2)$ for all ψ_1 and ψ_2;
4. $\mathrm{xnf}(\psi_1 \vee \psi_2) \equiv \mathrm{xnf}(\psi_1) \vee \mathrm{xnf}(\psi_2)$ for all ψ_1 and ψ_2;
5. $\mathrm{xnf}(\psi_1\,\mathcal{U}\,\psi_2) \equiv \mathrm{xnf}(\psi_2) \vee (\mathrm{xnf}(\psi_1) \wedge \mathsf{X}(\psi_1\,\mathcal{U}\,\psi_2))$ for all ψ_1 and ψ_2;
6. $\mathrm{xnf}(\psi_1\,\mathcal{R}\,\psi_2) \equiv \mathrm{xnf}(\psi_2) \wedge (\mathrm{xnf}(\psi_1) \vee \mathsf{X}(\psi_1\,\mathcal{R}\,\psi_2))$ for all ψ_1 and ψ_2.

Although the above definition has been recalled by other authors as well [17], it can be seen how it follows the same structure as the expansion rules defined in Table 2, which is not surprising, since these rules trace back to earlier graph-shaped tableaux [18, 19]. This connection makes it evident that the above definition produces an equivalent formula, as $\psi \equiv \Gamma_1(\psi) \vee \Gamma_2(\psi)$ for all the cases covered by Table 2.

3.2 Expansion of the tree

We can now define the first building block of our encoding. The *k-unraveling* of ϕ, denoted as $[\![\phi]\!]^k$, is a propositional formula that encodes the expansion of all the branches of the tableau tree up to at most $k+1$ poised nodes per branch.

Definition 2 (k-unraveling)**.** *Let ϕ be an LTL formula over Σ and some $k \in \mathbb{N}$. The* k-unraveling *of ϕ is a propositional formula $[\![\phi]\!]^k$ over Σ_+ defined as follows:*

$$
\begin{aligned}
[\![\phi]\!]^0 &= \mathrm{xnf}(\phi)_G \\
[\![\phi]\!]^{k+1} &= [\![\phi]\!]^k \wedge \textstyle\bigwedge_{\mathsf{X}\,\alpha \in XR} \left((\mathsf{X}\,\alpha)^k_G \leftrightarrow \mathrm{xnf}(\alpha)^{k+1}_G \right)
\end{aligned}
$$

Although such branches may in general have different length, they can be regarded as having the same depth as far as the corresponding model is concerned, since each state corresponds to a poised node. Thus, we may regard the k-unraveling as a symbolic encoding of a breadth-first traversal of the tree. The formula encodes the expansion rules by means of the next normal form transformation, and the STEP rule by tying the grounded X-request at step k with its (grounded) requested formula at step $k+1$, which ensures temporal consistency between two adjacent states in the model (*i.e.*, $\sigma \models_i \mathsf{X}\,\psi \leftrightarrow \sigma \models_{i+1} \psi$). Moreover, the CONTRADICTION rule is implicitly encoded as well, since satisfying assignments to the formula cannot represent branches containing propositional contradictions. Hence the following holds.

Proposition 1 (Soundness of the k-unraveling). *Let ϕ be an LTL formula. Then, $[\![\phi]\!]^k$ is unsatisfiable if and only if the complete tableau for ϕ contains only branches with at most $k+1$ poised nodes crossed by contradiction.* □

Note that $[\![\phi]\!]^{k+1}$ can be computed incrementally from $[\![\phi]\!]^k$, by adding only the second conjunct of the definition. This speeds up the construction of the formula itself as well as the solution process of modern incremental SAT-solvers.

3.3 Encoding of accepted branches

Once all non-crossed branches of a given depth have been identified with the k-unraveling, the *accepted* branches of such a depth can be represented by the conjunction of the propositional encoding of the EMPTY and LOOP rules of the tableau. This allows us to terminate the unraveling process in the case of a satisfiable formula.

The EMPTY rule, which is the simplest rule to encode, accepts *loop-free* models of the formula, that are identified by poised nodes lacking X-requests. In what follows, let $\mathsf{XR}_k \subseteq \mathsf{XR}$ be the set of X-requests that appear (grounded) in the k-th conjunct of the k-unraveling for ϕ. Similarly, let $\mathsf{XEV}_k \subseteq \mathsf{XR}_k$ be the X-eventualities (*i.e.*, formulae of the form $\mathsf{X}(\psi_1 \,\mathcal{U}\, \psi_2)$) found in XR_k. The EMPTY rule can be encoded as follows:

$$
E_k \coloneqq \bigwedge_{\varphi \in \mathsf{XR}_k} \neg \varphi^k_G
$$

Then, each satisfying assignment of the formula $[\![\phi]\!]^k \wedge E_k$ corresponds to a branch of the tableau for ϕ, with exactly $k+1$ poised nodes, accepted by the EMPTY rule. Note that it would still be sound to use the full XR instead of XR_k

in the definition above, but, in general, the latter is likely to be a smaller set, thus making the formula smaller.

The encoding of the LOOP rule, which accepts branches corresponding to *lasso-shaped* (periodic) models, is built on top of two pieces. For each $0 \leq l < k$, let ${}_lR_k$ and ${}_lF_k$ be defined as follows:

$$
\begin{aligned}
{}_lR_k &\coloneqq \bigwedge_{\psi \in \mathsf{XR}_k} \psi^l_G \leftrightarrow \psi^k_G \\
{}_lF_k &\coloneqq \bigwedge_{\substack{\psi \in \mathsf{XEV}_k \\ \psi \equiv \mathsf{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi^k_G \rightarrow \bigvee_{i=l+1}^{k} \mathrm{xnf}(\psi_2)^i_G \right)
\end{aligned}
$$

Given a branch $\overline{u} = \langle u_0, \ldots, u_k \rangle$ identified by $[\![\phi]\!]^k$, ${}_lR_k$ states that the nodes u_l and u_k have the same set of X-requests, and ${}_lF_k$ states that all such X-requests are fulfilled between nodes u_l and u_k. Together, they can be used to express the whole triggering condition of the LOOP rule:

$$
L_k \coloneqq \bigvee_{l=0}^{k-1} ({}_lR_k \wedge {}_lF_k)
$$

Then, each satisfying assignment of $[\![\phi]\!]^k \wedge L_k$ corresponds to a branch of the tableau for ϕ, with exactly $k{+}1$ poised nodes, accepted by the LOOP rule, *i.e.*, with a satisfying loop between position k and some previous position. Together, $[\![\phi]\!]^k$, E_k, and L_k can represent any accepted branch of the tableau of the given depth.

Definition 3 (Base encoding)**.** *Let ϕ be an LTL formula over Σ and $k \in \mathbb{N}$. The* base encoding *of ϕ at step k is the formula $|\phi|^k$ over Σ_+ defined as follows:*

$$
|\phi|^k \coloneqq \underbrace{[\![\phi]\!]^k}_{\substack{\textit{exp. rules} \\ \mathsf{STEP}\ \textit{rule}}} \wedge \left(\underbrace{E_k}_{\mathsf{EMPTY}\ \textit{rule}} \vee \underbrace{L_k}_{\mathsf{LOOP}\ \textit{rule}} \right)
$$

Again, note that the base encoding can be built incrementally, allowing us to exploit the features of modern SAT solvers. Indeed, $|\phi|^k$ consists of the conjunction of $[\![\phi]\!]^k$, built from the already computed $[\![\phi]\!]^{k-1}$, and $E_k \vee L_k$.

The construction of L_k gives us the following result.

Proposition 2 (Soundness of the base encoding). *Let ϕ be an LTL formula. Then, $|\phi|^k$ is satisfiable if and only if the complete tableau for ϕ contains at least an accepted branch with exactly $k + 1$ poised nodes.* □

Propositions 1 and 2, together with the soundness result for Reynolds' tableau given in [22], lead us to the following result.

Theorem 1 (Soundness). *Given an LTL formula ϕ, if $|\phi|^k$ is satisfiable, for some $k \in \mathbb{N}$, then ϕ is satisfiable.* □

```
procedure BSC(ϕ)
    k ← 0
    while True do
        generate |ϕ|^k
        if |ϕ|^k is SAT then
            ϕ is SAT
            stop
        k ← k + 1
```

Fig. 1. Incomplete satisfiability checking procedure built on top of the base encoding.

Figure 1 shows a basic procedure that can be built on top of the encoding of Definition 3. The procedure starts with $k = 0$, and increments it at each step, looking for models of increasing size, stopping when a step k is found with a satisfiable base encoding. The procedure is incomplete, as it may not terminate on unsatisfiable instances, similarly to early *bounded model checking* techniques.

If the procedure terminates, then the satisfying assignment for $|\phi|^k$ can be used to build a model $\sigma \subseteq \Sigma^\omega$ of ϕ of minimal length, where, in the case of periodic models, the length is considered as the sum of the prefix and the period lengths. This breadth-first traversal, with the guarantee of finding a minimal model, would not be feasible if carried out explicitly, and it is a distinguishing feature of bounded satisfiability checking of this kind. Explicit implementations of Reynolds' tableau system [3] proceed instead in a depth-first way, and the models they find are *not* guaranteed to be minimal in length.

The next section adds to the picture the encoding of the PRUNE rule, showing how to integrate the above procedure in order to guarantee the termination for any unsatisfiable instance as well.

4 Completeness

In order to ensure termination of the algorithm in Figure 1 also on unsatisfiable formulae, it is useful to look at the possible reasons why the base encoding $|\phi|^k$ of a formula ϕ may be unsatisfiable. We can distinguish two cases:

1. if the formula $[\![\phi]\!]^k$ is unsatisfiable, it means that all the branches of the tableau for ϕ are crossed by the CONTRADICTION rule at or before depth k (see Proposition 1);
2. if both $[\![\phi]\!]^k \wedge E_k$ and $[\![\phi]\!]^k \wedge L_k$ are unsatisfiable, then there are no branches of depth k accepted by the EMPTY rule or by the LOOP rule (see Proposition 2).

As an example of the first case, consider the formula $\mathsf{X}\, p \wedge \mathsf{X}\, \neg p$, whose 1-unraveling is $[\![\mathsf{X}\, p \wedge \mathsf{X}\, \neg p]\!]^1 \equiv (\mathsf{X}\, p)^0_G \wedge (\mathsf{X}\, \neg p)^0_G \wedge p^1 \wedge \neg p^1$. At step $k = 1$, the formula is found to be unsatisfiable because of a propositional contradiction between p^1 and $\neg p^1$. At this point there is no reason to continue looking further: we can stop incrementing k and answer UNSAT.

The second case, instead, does *not* exclude that longer accepted branches exist, and require looking further. One interesting example is the (unsatisfiable) formula $\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p$: it holds that $|\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p|^k$ is unsatisfiable for all $k \geq 0$, since any branch can be accepted neither by the LOOP rule (because $\mathsf{G}\,\neg p$ forces p^i to be false for each $0 \leq i \leq k$) nor by the EMPTY rule (because the failed fulfilment of $q\,\mathcal{U}\,p$ forces $\mathsf{X}(q\,\mathcal{U}\,p)^i$ to be true for each $0 \leq i \leq k$). Nevertheless, $[\![\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p]\!]^k$ is satisfiable for all $k \geq 0$, because the branch of the tableau that indefinitely postpones the satisfaction of $q\,\mathcal{U}\,p$ is never closed by contradiction. Hence, the procedure in Figure 1 can never be able to stop in this case.

In the tableau, such a branch is, instead, rejected by the PRUNE rule, whose role is exactly that of rejecting these potentially infinite branches. We can similarly recover termination and completeness of our procedure by introducing a propositional encoding of the rule.

Recall that the PRUNE rule rejects any branch of length k that presents two positions $l < j < k$, with the same set of X-requests, such that all the X-eventualities fulfilled between $j+1$ and k are fulfilled between $l+1$ and j as well. Let i and j be one such pair of positions. We can encode the condition of the PRUNE rule by means of the following formula:

$$ {}_lP_j^k \coloneqq \bigwedge_{\substack{\psi \in \mathsf{XEV}_k \\ \psi \equiv \mathsf{X}(\psi_1 \mathcal{U} \psi_2)}} \left(\psi_G^k \wedge \bigvee_{i=j+1}^{k} \mathrm{xnf}(\psi_2)_G^i \to \bigvee_{i=l+1}^{j} \mathrm{xnf}(\psi_2)_G^i \right) $$

Then, the above formula can be combined with the ${}_lR_k$ formula defined in the previous section to obtain the following encoding of the PRUNE rule:

$$ P^k \coloneqq \bigvee_{l=0}^{k-2} \bigvee_{j=l+1}^{k-1} \left({}_lR_j \wedge {}_jR_k \wedge {}_lP_j^k \right) $$

It is worth to note that the P^k formula is of cubic size with respect to k *and* the number of X-eventualities. With this formula, in case of an unsatisfiable base encoding, we can check whether there exists at least one branch of depth at most k which does *not* satisfy the prune condition: if this is the case, then it makes sense to continue the search; otherwise, the procedure can stop reporting the unsatisfiability of the formula. This is done by testing the satisfiability of the *termination encoding* of ϕ, defined as the following formula:

$$ |\phi|_T^k \coloneqq \underbrace{[\![\phi]\!]^k}_{\substack{\text{exp. rules} \\ \textsf{STEP}\text{ rule}}} \wedge \bigwedge_{i=0}^{k} \underbrace{\neg P^i}_{\textsf{PRUNE}\ \textit{rule}} $$

The complete procedure is shown in Figure 2, where the first step k such that $|\phi|_T^k$ is unsatisfiable stops the search. Based on the soundness and completeness result for the encoded tableau system [22], we can state the following result.

Theorem 2 (Soundness and completeness). *For every LTL formula ϕ, the procedure of Figure 2 always terminates, and it answers SAT iff ϕ is satisfiable.*

1: **procedure** LTL-SAT-PRUNE(ϕ)
2:     $k \leftarrow 0$
3:     **while** True **do**
4:         generate $[\![\phi]\!]^k$
5:         **if** $[\![\phi]\!]^k$ is UNSAT **then**
6:             ϕ is UNSAT
7:             stop
8:         generate $|\phi|^k$
9:         **if** $|\phi|^k$ is SAT **then**
10:             ϕ is SAT
11:             stop
12:         generate $|\phi|_T^k$
13:         **if** $|\phi|_T^k$ is UNSAT **then**
14:             ϕ is UNSAT
15:             stop
16:         $k \leftarrow k + 1$

Fig. 2. Complete and terminating satisfiability checking procedure based on the tableau encoding

Notably, the procedure guarantees termination and completeness without establishing *a priori* a bound to the depth of the tree, at the cost of a slightly bigger formula and three calls to the underlying solver.

It is worth to spend some words on how the above procedure can exploit the *incrementality* of modern SAT solvers to speed up its execution. Many modern solvers have a push/pop interface that allows the client to push some conjuncts to a stack, solve them, then pop some of them while pushing others, maintaining all the information about the untouched conjuncts. In our case, the construction of $[\![\phi]\!]^k$ only requires the addition of a conjunct to $[\![\phi]\!]^{k-1}$, and $|\phi|^k$ only requires to join $E_k \vee L_k$ to $[\![\phi]\!]^k$. This means that such a conjunct can be pushed temporarily, while maintaining all the solver state about $[\![\phi]\!]^k$ for the next step. Moreover, the formula $[\![\phi]\!]^k$ generated and solved at Lines 4 and 5 of Figure 2 can be replaced by one built on top of the whole $|\phi|_T^{k-1}$ from the previous step, instead of only from $[\![\phi]\!]^{k-1}$. This allows us to avoid to backtrack the additional conjuncts of $|\phi|_T^k$. Since the PRUNE rule cuts redundant branches, maintaining the corresponding formulae from step to step helps guiding the solver through relevant branches.

5 Experimental evaluation

The procedure described in this paper is implemented in a tool called BLACK (Bounded Ltl sAtisfiability ChecKer).[3] This section describes relevant aspects of the tool and shows the results of our preliminary experimental evaluation, where it has been compared with other state-of-the-art LTL solvers.

[3] BLACK can be downloaded from `https://github.com/black-sat/black`, together with the whole benchmarking suite and the raw results data.

BLACK has been implemented from scratch in the C++17 language with the goals of efficiency, portability, and reusability. Most of the tool is implemented as a shared library with a well-defined API, that can be linked to other client applications as needed. The library provides basic formula handling facilities, and an interface to the main solving algorithm. The tool itself is as well a client of such a library, providing a simple command-line user interface.

The tool is currently implemented on top of MathSAT [5], used as its back-end SAT solver, which is actually a full-blown SMT solver. This choice was driven by the fact that, contrary to most pure-SAT solvers, MathSAT supports formulae with a general syntax, without the need of a preliminary conversion to CNF. This feature greatly simplified the initial development cycle of the project. Future plans include the support to multiple different SAT solvers, including those with simple CNF-based APIs, to find the most performant candidate.

The satisfiability checking procedure described above is implemented on top of a formula handling layer, which eases the development of the solver by decoupling the logical encoding from low-level details. In particular, the lower layer transparently implements *subterm sharing*, *i.e.*, formulas are internally represented as *circuits*, by identifying repeated subformulas. Besides the positive effects on memory usage, this mechanism matches well with the term-based API of the MathSAT library. Most importantly, *syntactic equality* of two formulae reduces to a single pointer comparison, since building any two equal formulae results into two pointers to the same object. A peculiar feature of BLACK's formulas handling layer is that atomic propositions can be labeled by values of almost any data type, in contrast to being restricted to strings, integers, or similar identifiers. In this way, the *grounding* operation (ψ_G) performed on X-requests by our encoding (such as in $[\![\phi]\!]^k$) is effectively a *no-op*: the grounding of an X-request formula is just an atomic proposition labelled by the formula's representing object, with no need for any translation table between the formulae and their corresponding grounded symbols. Since formulas are uniquely identified by just the pointer to their object, this is implementable in such a way that the common cases of propositions labeled by short strings, formulae, and formula/integer pairs (for the *stepped* versions ψ_G^k) do not cause any memory allocation.

In our experiments, we compared BLACK with four competitors: *Aalta* v2.0 [17], *nuXmv* [6], *Leviathan* [3], and *PLTL* [1,24]. The *nuXmv* model checker is tested in two modes, which implement, respectively, the *Simple Bounded Model Checking* (SBMC) [15] and the *K-Liveness* [8] techniques. The SBMC mode is the most similar to ours among the tested solvers. The *PLTL* tool implements both a graph-shaped [1] and an almost tree-shaped [24] tableau techniques. Finally, *Leviathan* is an explicit implementation of Reynolds' tableau [3]. Because of technical issues, we could not include the LS4 [26] tool in our test. Future experiments will include this and other competitors as well.

We considered the comprehensive set of formulae collected by Schuppan and Darmawan [23], which contains a total of 3723 LTL formulae, grouped in seven families, `acacia`, `alaska`, `anzu`, `forobots`, `rozier`, `schuppan`, `trp`, named after their original source. We set a timeout of five minutes for each formula in the set.

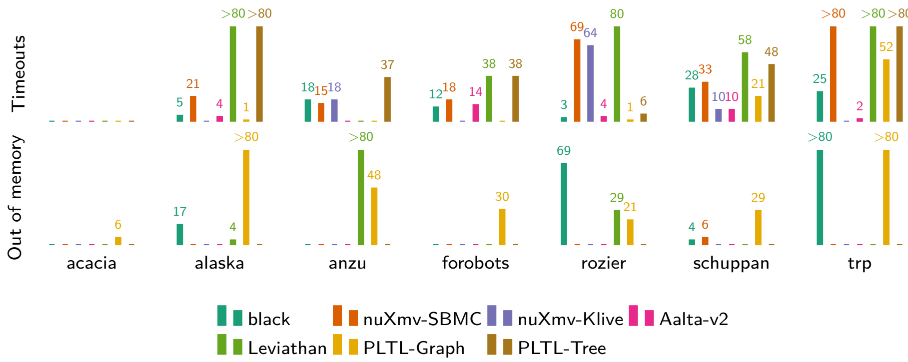

Fig. 3. Total number of *timeouts* and *out of memory* interruptions of the solvers on the different class of benchmark formulas.

We ran our tests on a Quad Core i5-2500k 3.30GHz processor, with 8GB of main memory. Processes were assigned a single CPU core each, with a memory limit of 2GB per core and a five minutes timeout. Figures 4 and 5 show six scatter plots comparing the execution times, while Figure 3 shows the number of *timeouts* and *out of memory* interruptions for the tools on each class of formulas.

Overall, the results are promising. Although *Aalta* remains the most performant tool, the picture is mixed. In particular, BLACK is competitive with regards to *nuXmv*. With regards to the SBMC mode, the advantage is consistent but constant, showing similar trends both on satisfiable and unsatisfiable instances. The `rozier` set comes as an exception. Apart from the *counter* formulas, which are hard for both solvers, all these formulas have very short models, which is an advantage for iterative deepening approaches like ours. SBMC shares the same principle, but the large difference between BLACK and *nuXmv* on most of this set may be explained by (i) the simpler base encoding employed by BLACK, whose asymptotically larger size does not bite at lower values of the bound k, and/or (ii) differences between the SAT solvers underlying the two tools (the distributed binary of *nuXmv* is linked to *minisat* [12]). When comparing with *nuXmv* in *k-liveness* mode, we can see an interesting pattern on `trp` unsatisfiable instances, with some formulas being solved in milliseconds while others reach the timeout limit. As recalled in [23], this is a set of *random* instances, hence the erratic behavior cannot *a priori* be tied to any particular combination of parameters. The comparison with *Leviathan* and the other explicit tableaux implemented by *PLTL* is easier to analyze. BLACK performs consistently better than the two tools, which suffer from a predictable explosion in memory usage in most instances. Notably, they perform very well on formulas with very narrow search trees, such as the `rozier` *counters*.

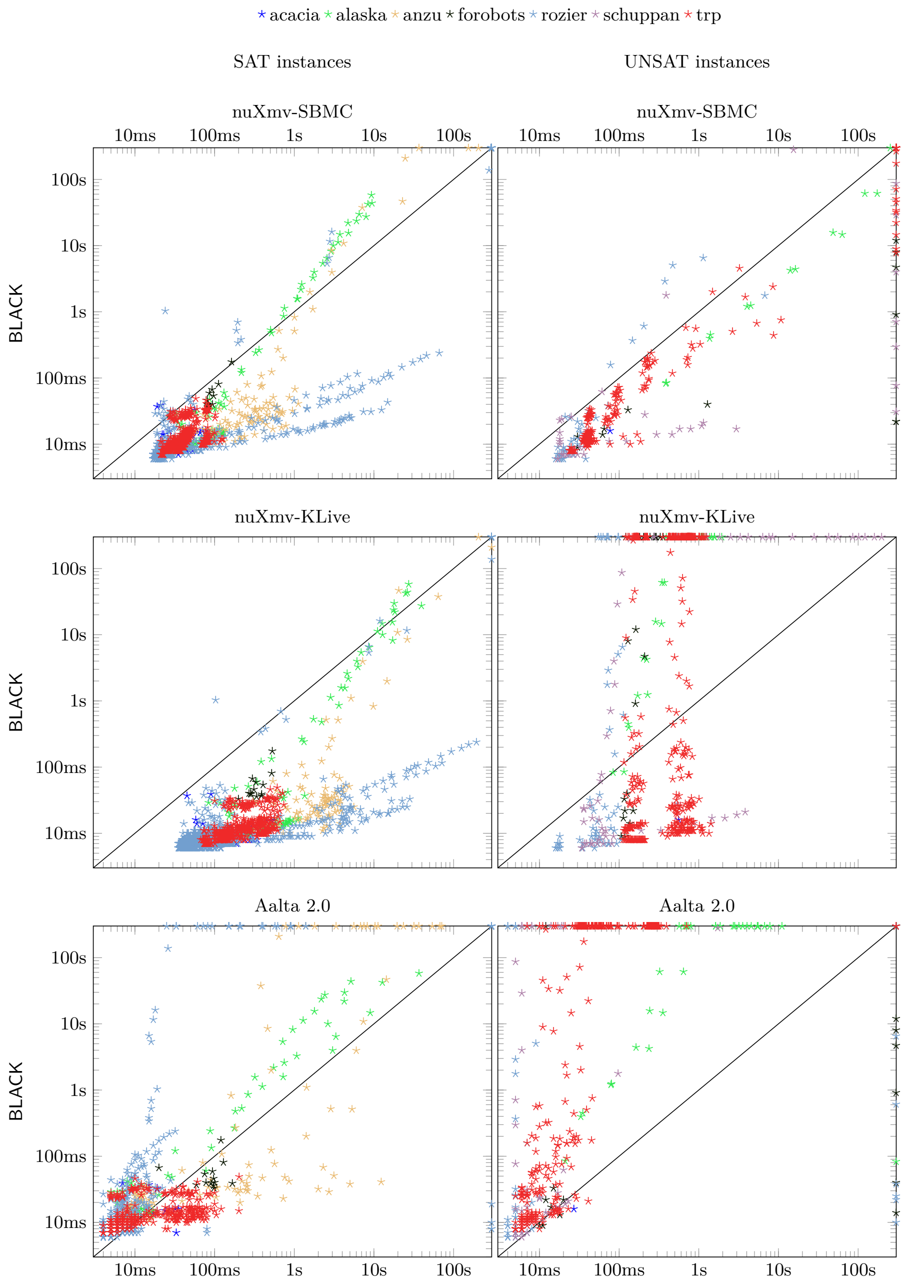

Fig. 4. Experimental comparison with *nuXmv* and *Aalta*

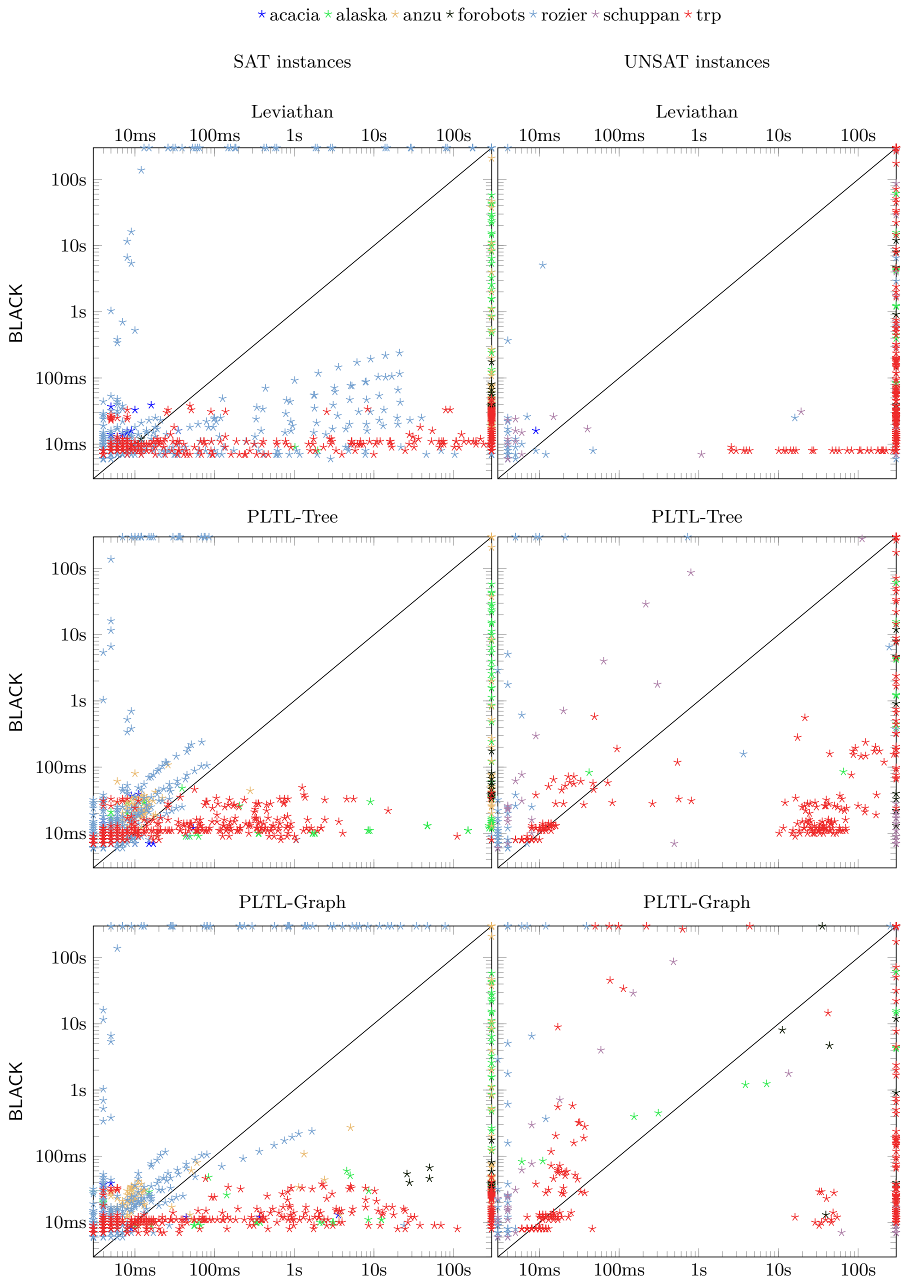

Fig. 5. Experimental comparison with *Leviathan* and *PLTL*

6 Conclusions

This paper described a satisfiability checking algorithm for LTL formulae based on a SAT encoding of Reynolds' one-pass and tree-shaped tableau system [22]. Both the expansion of the tableau tree and its rules are represented by Boolean formulae, whose satisfying assignments represent all the branches of the tableau up to a given depth k. Notably, the encoding of Reynolds' PRUNE rule results in a simple yet effective termination condition for the algorithm, which is a non-trivial task in other *bounded model checking* approaches [15].

We implemented our procedure in the BLACK tool and made some preliminary experimental comparison with state-of-the-art LTL solvers. The tool shows good performance overall. In particular, it outperforms *Leviathan*, the *explicit* implementation of Reynolds' tableau, and shows interesting results against the similar *simple bounded model checking* approach. The results are promising, especially considering that the encoding has been implemented very simply as shown above, without any sort of heuristics in the generation of the encoded formulas. Further work should consider more compact encoding for the unraveling and for the LOOP and PRUNE rules, the use and comparison of different backend SAT solvers, and heuristics for the search of the bound.

From a theoretical perspective, the followed approach has to be compared with others, especially with *bounded* ones [15], on a conceptual, rather than experimental, level. In particular, it is worth comparing the PRUNE rule with the terminating conditions exploited in other bounded approaches, to understand their difference and draw possible connections.

A number of extensions of Reynolds' tableau to other logics have been proposed since its inception. In particular, the extension to past operators [14] appears to be easy to be encoded, without resorting to the *virtual unrollings* technique used in other bounded approaches [15]. Reynolds' tableau system has also been extended to timed logics [13], in particular TPTL [2] and a TPTL_b+P [11]. It is natural to ask whether the approach used here to encode the LTL tableau to SAT can be adapted to encode the timed extensions of the tableau to SMT.

Acknowledgments

This work has been supported by the PRID project *ENCASE - Efforts in the uNderstanding of Complex interActing SystEms*, and by the INdAM GNCS project *Formal Methods for Combined Verification.* The authors thank *Alessandro Cimatti* and *Stefano Tonetta* for the helpful discussions about bounded satisfiability checking, and *Valentino Picotti* for providing the benchmarking hardware.

References

1. Abate, P., Goré, R., Widmann, F.: An On-the-fly Tableau-based Decision Procedure for PDL-satisfiability. Electronic Notes in Theoretical Computer Science **231**, 191–209 (2009). https://doi.org/10.1016/j.entcs.2009.02.036
2. Alur, R., Henzinger, T.A.: A Really Temporal Logic. Journal of the ACM **41**(1), 181–204 (1994)
3. Bertello, M., Gigante, N., Montanari, A., Reynolds, M.: Leviathan: A new LTL satisfiability checking tool based on a one-pass tree-shaped tableau. In: Proc. of the 25$^{\text{th}}$ International Joint Conference on Artificial Intelligence. pp. 950–956. IJCAI/AAAI Press (2016)
4. Beth, E.W.: Semantic entailment and formal derivability. Sapientia **14**(54), 311 (1959)
5. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The mathsat 4 smt solver. In: International Conference on Computer Aided Verification. pp. 299–303. Springer (2008)
6. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: International Conference on Computer Aided Verification. pp. 334–342. Springer (2014)
7. Cimatti, A., Roveri, M., Sheridan, D.: Bounded verification of past LTL. In: International Conference on Formal Methods in Computer-Aided Design. pp. 245–259. Springer (2004)
8. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: 2012 Formal Methods in Computer-Aided Design (FMCAD). pp. 52–59. IEEE (2012)
9. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (2001)
10. D'Agostino, M., Gabbay, D., Hähnle, R., Posegga, J. (eds.): Handbook of Tableau Methods. Springer (1999)
11. Della Monica, D., Gigante, N., Montanari, A., Sala, P., Sciavicco, G.: Bounded timed propositional temporal logic with past captures timeline-based planning with bounded constraints. In: Proc. of the 26$^{\text{th}}$ International Joint Conference on Artificial Intelligence. pp. 1008–1014 (2017). https://doi.org/10.24963/ijcai.2017/140
12. Eén, N., Sörensson, N.: An extensible sat-solver. In: Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers. pp. 502–518 (2003). https://doi.org/10.1007/978-3-540-24605-3_37, `https://doi.org/10.1007/978-3-540-24605-3_37`
13. Geatti, L., Gigante, N., Montanari, A., Reynolds, M.: One-pass and tree-shaped tableau systems for TPTL and TPTLb+Past. In: Orlandini, A., Zimmermann, M. (eds.) Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018. EPTCS, vol. 277, pp. 176–190 (2018). https://doi.org/10.4204/EPTCS.277.13, `https://doi.org/10.4204/EPTCS.277.13`
14. Gigante, N., Montanari, A., Reynolds, M.: A one-pass tree-shaped tableau for LTL+Past. In: Proc. of 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning. EPiC Series in Computing, vol. 46, pp. 456–473 (2017)
15. Heljanko, K., Junttila, T., Latvala, T.: Incremental and complete bounded model checking for full PLTL. In: International Conference on Computer Aided Verification. pp. 98–111. Springer (2005)

16. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE transactions on robotics **25**(6), 1370–1381 (2009)
17. Li, J., Zhu, S., Pu, G., Vardi, M.Y.: Sat-based explicit LTL reasoning. In: Haifa Verification Conference. pp. 209–224. Springer (2015)
18. Lichtenstein, O., Pnueli, A.: Propositional Temporal Logics: Decidability and Completeness. Logic Journal of the IGPL **8**(1), 55–85 (2000). https://doi.org/10.1093/jigpal/8.1.55
19. Manna, Z., Pnueli, A.: Temporal Verification of Reactive Systems - Safety. Springer (1995)
20. Mayer, M.C., Limongelli, C., Orlandini, A., Poggioni, V.: Linear temporal logic as an executable semantics for planning languages. Journal of Logic, Language and Information **16**(1), 63–89 (2007). https://doi.org/10.1007/s10849-006-9022-1
21. McCabe-Dansted, J.C., Reynolds, M.: A parallel linear temporal logic tableau. In: Bouyer, P., Orlandini, A., Pietro, P.S. (eds.) Proceedings 8$^{\text{th}}$ International Symposium on Games, Automata, Logics and Formal Verification. EPTCS, vol. 256, pp. 166–179 (2017)
22. Reynolds, M.: A New Rule for LTL Tableaux. In: Proc. of the 7$^{\text{th}}$ International Symposium on Games, Automata, Logics and Formal Verification. EPTCS, vol. 226, pp. 287–301 (2016). https://doi.org/10.4204/EPTCS.226.20
23. Schuppan, V., Darmawan, L.: Evaluating LTL Satisfiability Solvers. In: Proc. of the 9$^{\text{th}}$ International Symposium on Automated Technology for Verification and Analysis. pp. 397–413 (2011)
24. Schwendimann, S.: A new one-pass tableau calculus for PLTL. In: Proc. of the 7$^{\text{th}}$ International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. LNCS, vol. 1397, pp. 277–292. Springer (1998). https://doi.org/10.1007/3-540-69778-0_28
25. Sistla, A.P., Clarke, E.M.: The Complexity of Propositional Linear Temporal Logics. Journal of ACM **32**(3), 733–749 (1985). https://doi.org/10.1145/3828.3837
26. Suda, M., Weidenbach, C.: A PLTL-Prover Based on Labelled Superposition with Partial Model Guidance. In: Proc. of the 6$^{\text{th}}$ International Joint Conference on Automated Reasoning. LNCS, vol. 7364, pp. 537–543. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_42
27. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings fifth ieee international symposium on requirements engineering. pp. 249–262. IEEE (2001)